data_derived_narratives

May 6, 2020

[1]:
```
import geopandas as gpd
import pandas as pd
import numpy as np
from shapely.geometry import Point, Polygon
import random
from fiona.crs import from_epsg
```

[2]:
```
import urllib, json, requests, geojson
import googlemaps
```

[3]:
```
from ipyleaflet import (
    Map, GeoData, GeoJSON, basemaps, basemap_to_tiles,
    Icon, Circle, Marker,
    LayerGroup, WidgetControl)
import ipywidgets as widgets
from ipywidgets import Button, Layout
from IPython.display import display, clear_output, Markdown as md
```

[4]:
```
output = widgets.Output()
data_output = widgets.Output()
```

[5]:
```
#need this to stop numpy from returning truncated arrays
import sys
np.set_printoptions(threshold=sys.maxsize)

# for automatic linebreaks and multi-line cells
pd.options.display.max_colwidth = 10000
```

[6]:
```
center = (40.7210907,-73.9877836)
basemap = basemap_to_tiles(basemaps.CartoDB.Positron)

m = Map(layers=(basemap, ), center=center, zoom=14, min_zoom = 7, max_zoom = 20)
```

[7]:
```
def extract_location():
    global gdf, lat, lon

    lat = str(markerlocation[0])
```

```
    lon = str(markerlocation[1])

    df2 = pd.DataFrame(markerlocation)
    df=df2.transpose()
    df.columns=['Latitude','Longitude']

    gdf = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df.Longitude, df.
 ↪Latitude), crs='epsg:4326')

    return gdf
```

[8]:
```
draggable=False
marker_opacity=0
icon = Icon(icon_url='icon.png', icon_size=[15, 15])

marker = Marker(location=center, draggable=draggable, icon=icon,␣
 ↪opacity=marker_opacity)
studyarea = Circle(location=center, radius=420, color="#F0C677",␣
 ↪fill_color="#f2eecb", fill_opacity = .4, weight=2)

markerlocation = marker.location

layer_group = LayerGroup(layers=(marker, studyarea))
m.add_layer(layer_group)

def update_marker(**kwargs):

    if kwargs.get('type') == 'click':
        layer_group.clear_layers();

        global studyarea

        marker = Marker(location=kwargs.get('coordinates'),␣
 ↪draggable=draggable, icon=icon, opacity=marker_opacity,␣
 ↪options=['rise_on_hover'])
        studyarea = Circle(location=kwargs.get('coordinates'), radius=420,␣
 ↪color="#F0C677", fill_color="#f2eecb", fill_opacity = .4, weight=2)

        global markerlocation
        markerlocation = marker.location

        layer_group.add_layer(marker)
        layer_group.add_layer(studyarea)

        draw_update_buffer(**kwargs)

        output.clear_output()
```

[9]:
```
def draw_update_buffer(**kwargs):
    m.on_interaction(update_marker)
    extract_location()

    global half_mi
    half_mi=gdf.copy()
    half_mi['geometry'] = half_mi.geometry.buffer(.004,  cap_style=1,␣
 ↪join_style=1)

draw_update_buffer()
```

[10]:
```
def import_censustracts():
    extract_location()

    global bounds
    bounding_box = half_mi.envelope
    df = gpd.GeoDataFrame(gpd.GeoSeries(bounding_box), columns=['geometry'])
    minx, miny, maxx, maxy = df.geometry.total_bounds
    bounds = minx, miny, maxx, maxy

    # census tracts link
    endpoint = 'https://tigerweb.geo.census.gov/arcgis/rest/services/TIGERweb/
 ↪Tracts_Blocks/MapServer/4/query'
    s = requests.session()
    s.params = {
        'geometry': str(bounds),
        'geometryType': 'esriGeometryEnvelope',
        'inSR': 4326,
        'spatialRel': 'esriSpatialRelIntersects',
        'outFields': 'GEOID,STATE,COUNTY,TRACT,NAME,STGEOMETRY,OBJECTID',
        'returnGeometry': True,
        'f': 'geojson',
    }
    start = 0
    done = False
    features = []
    crs = None
    while not done:
        r = s.get(endpoint, params={
            'resultOffset': start,
            'resultRecordCount': 32,
        })
        censusgeo = geojson.loads(r.text)
        newfeats = censusgeo.__geo_interface__['features']
        if newfeats:
            features.extend(newfeats)
            crs=censusgeo.__geo_interface__['crs']
```

```
            start += len(newfeats)
        else:
            done = True

    global tracts
    tracts = gpd.GeoDataFrame.from_features(features, crs=crs)
```

[11]:
```
def download_acs():
    global tract, state, county

    state = tracts["STATE"].unique().tolist()
    state = ', '.join(map(str, state)).replace(" ", "")

    tract = tracts["TRACT"].unique().tolist()
    tract = ', '.join(map(str, tract)).replace(" ", "")

    county = tracts["COUNTY"].unique().tolist()
    county = ', '.join(map(str, county)).replace(" ", "")

    api_key = '9330dc4bf086a84f19fb412bb15f232507301de6'
    acs_url = f'https://api.census.gov/data/2018/acs/acs5/subject/'

    global acs_variables
    acs_variables_initial =␣
↪'S1603_C02_002E,S1603_C02_003E,S1603_C02_004E,S1603_C04_002E,S1603_C04_003E,S1603_C04_004E,
    acs_variables_additional =␣
↪'S1501_C01_002E,S1501_C01_004E,S1501_C01_003E,S1501_C01_005E,S1501_C01_017E,S1501_C01_018E,
    acs_variables = acs_variables_initial + "," + acs_variables_additional

    get_acs_initial = f'{acs_url}?&get={acs_variables_initial}&for=tract:
↪{tract}&in=state:{state}%20county:{county}&key={api_key}'
    get_acs_additional = f'{acs_url}?&get={acs_variables_additional}&for=tract:
↪{tract}&in=state:{state}%20county:{county}&key={api_key}'

    data_acs_initial=requests.get(get_acs_initial).json()
    data_acs_additional=requests.get(get_acs_additional).json()

    global acs
    acs_initial=pd.DataFrame(data_acs_initial[1:], columns=data_acs_initial[0])
    acs_additional=pd.DataFrame(data_acs_additional[1:],␣
↪columns=data_acs_additional[0])

    acs=pd.merge(acs_initial, acs_additional, on='tract', how='left')
```

[12]:
```
def clean_combine_census_and_geographic_data():
    import_censustracts()
```

```
    download_acs()

    global acs_site_sum, acs_site
    tracts["area"]=tracts.area
    acs_tracts = pd.merge(tracts, acs, left_on='TRACT', right_on='tract',␣
↪how='left')

    acs_site = gpd.overlay(half_mi, acs_tracts, how='intersection')
    acs_site["area_clipped"]=acs_site.area
    acs_site["ratio"] = acs_site["area_clipped"]/acs_site["area"]

    cols = acs_variables.split(",")
    acs_site[cols] = acs_site[cols].apply(pd.to_numeric, errors='coerce',␣
↪axis=1)

    temp_df = acs_site[cols]
    temp_df = temp_df.mul(acs_site.ratio, 0)
    acs_site.update(temp_df)

    acs_site_sum = pd.DataFrame(acs_site[cols].sum())

    acs_site_sum.reset_index(inplace=True)
    acs_site_sum.columns = ['variables', 'sum_in_area']
```

[13]:
```
def import_metropolitan_statistical_areas():
    endpoint_metropolitan = 'https://tigerweb.geo.census.gov/arcgis/rest/
↪services/TIGERweb/CBSA/MapServer/8/query'
#     endpoint_MSAs='https://tigerweb.geo.census.gov/arcgis/rest/services/
↪TIGERweb/CBSA/MapServer/6/query'

    s = requests.session()
    s.params = {
        'geometry': str(bounds),
        'geometryType': 'esriGeometryEnvelope',
        'inSR': 4326,
        'spatialRel': 'esriSpatialRelIntersects',
        'outFields': 'GEOID,CBSA,NAME,STGEOMETRY,OBJECTID',
        'returnGeometry': True,
        'f': 'geojson',
    }
    start = 0
    done = False
    features = []
    crs = None
    while not done:
        r = s.get(endpoint_metropolitan, params={
            'resultOffset': start,
```

```
            'resultRecordCount': 32,
        })
        censusgeo = geojson.loads(r.text)
        newfeats = censusgeo.__geo_interface__['features']
        if newfeats:
            features.extend(newfeats)
            crs=censusgeo.__geo_interface__['crs']
            start += len(newfeats)
        else:
            done = True

    global metropolitan
    metropolitan = gpd.GeoDataFrame.from_features(features, crs=crs)
```

[14]:
```
def import_micropolitan_statistical_areas():
    endpoint_micropolitan = 'https://tigerweb.geo.census.gov/arcgis/rest/
↪services/TIGERweb/CBSA/MapServer/9/query'

    s = requests.session()
    s.params = {
        'geometry': str(bounds),
        'geometryType': 'esriGeometryEnvelope',
        'inSR': 4326,
        'spatialRel': 'esriSpatialRelIntersects',
        'outFields': 'GEOID,CBSA,NAME,STGEOMETRY,OBJECTID',
        'returnGeometry': True,
        'f': 'geojson',
    }
    start = 0
    done = False
    features = []
    crs = None
    while not done:
        r = s.get(endpoint_micropolitan, params={
            'resultOffset': start,
            'resultRecordCount': 32,
        })
        censusgeo = geojson.loads(r.text)
        newfeats = censusgeo.__geo_interface__['features']
        if newfeats:
            features.extend(newfeats)
            crs=censusgeo.__geo_interface__['crs']
            start += len(newfeats)
        else:
            done = True

    global micropolitan
```

```
    micropolitan = gpd.GeoDataFrame.from_features(features, crs=crs)
```

[15]:
```
def get_msas():
    import_metropolitan_statistical_areas()
    import_micropolitan_statistical_areas()

    global msa
    if ((len(metropolitan.index) > 0) & (len(micropolitan.index) > 0)):
        msa = pd.merge(metropolitan, micropolitan, on='GEOID', how='left')
    elif ((len(metropolitan.index) > 0) & (len(micropolitan.index) == 0)):
        msa = metropolitan.copy()
    elif ((len(metropolitan.index) == 0) & (len(micropolitan.index) > 0)):
        msa = micropolitan.copy()

    msa["area"]=msa.area
```

[16]:
```
def get_worker_and_resident_populations():
    get_msas()

    api_key = '9330dc4bf086a84f19fb412bb15f232507301de6'
    acs_url = f'https://api.census.gov/data/2018/acs/acs5/subject/'
    qwi_url = f'https://api.census.gov/data/timeseries/qwi/sa/'

    global acs_population
    acs_pop_variable = 'S0101_C01_001E'
    qwi_pop_variable = 'EmpS'
    MSA_short='metropolitan%20statistical%20area/
↪micropolitan%20statistical%20area'
    time='2019-Q2'

    get_acs_pop = f'{acs_url}?&get={acs_pop_variable}&for=tract:
↪{tract}&in=state:{state}%20county:{county}&key={api_key}'
    get_qwi_pop = f'{qwi_url}?&get={qwi_pop_variable}&for={MSA_short}:
↪*&for=county:*&in=state:{state}&time={time}&key={api_key}'

    data_acs_pop=requests.get(get_acs_pop).json()
    data_qwi_pop=requests.get(get_qwi_pop).json()

    global acs_pop, qwi_pop, qwi_pop_site_sum, acs_pop_site_sum
    acs_pop=pd.DataFrame(data_acs_pop[1:], columns=data_acs_pop[0])
    qwi_pop=pd.DataFrame(data_qwi_pop[1:], columns=data_qwi_pop[0])

    acs_pop_tracts = pd.merge(tracts, acs_pop, left_on='TRACT',␣
↪right_on='tract', how='left')
    acs_pop_site = gpd.overlay(half_mi, acs_pop_tracts, how='intersection')
    acs_pop_site["area_clipped"]=acs_pop_site.area
    acs_pop_site["ratio"] = acs_pop_site["area_clipped"]/acs_pop_site["area"]
```

```
    acs_pop_site[acs_pop_variable] = acs_pop_site[acs_pop_variable].apply(pd.
↪to_numeric, errors='coerce')
    temp_df_acs = acs_pop_site[acs_pop_variable]
    temp_df_acs = temp_df_acs.mul(acs_pop_site.ratio, 0)
    acs_pop_site.update(temp_df_acs)
    acs_pop_site_sum = pd.DataFrame(acs_pop_site[[acs_pop_variable]].sum())
    acs_pop_site_sum.reset_index(inplace=True)
    acs_pop_site_sum.columns = ['variables', 'sum_in_area']


    qwi_pop_msa = pd.merge(msa, qwi_pop, left_on='GEOID',␣
↪right_on='metropolitan statistical area/micropolitan statistical area',␣
↪how='left')
    qwi_pop_site = gpd.overlay(half_mi, qwi_pop_msa, how='intersection')
    qwi_pop_site["area_clipped"]=qwi_pop_site.area
    qwi_pop_site["ratio"] = qwi_pop_site["area_clipped"]/qwi_pop_site["area"]
    qwi_pop_site[qwi_pop_variable] = qwi_pop_site[qwi_pop_variable].apply(pd.
↪to_numeric, errors='coerce')
    temp_df_qwi = qwi_pop_site[qwi_pop_variable]
    temp_df_qwi = temp_df_qwi.mul(qwi_pop_site.ratio, 0)
    qwi_pop_site.update(temp_df_qwi)
    qwi_pop_site_sum = pd.DataFrame(qwi_pop_site[[qwi_pop_variable]].sum())
    qwi_pop_site_sum.reset_index(inplace=True)
    qwi_pop_site_sum.columns = ['variables', 'sum_in_area']
```

[17]:
```
def determine_population_distribution():
    get_worker_and_resident_populations()

    residents = int(acs_pop_site_sum.sum_in_area)
    workers = int(qwi_pop_site_sum.sum_in_area)
    total_pop = residents + workers

    global resident_percent, worker_percent, more_residents, more_workers
    resident_percent = int((residents/total_pop)*100)
    worker_percent = int((workers/total_pop)*100)

    if resident_percent > worker_percent:
        more_residents = resident_percent - worker_percent
        more_workers = 0
    elif resident_percent < worker_percent:
        more_workers = worker_percent - resident_percent
        more_residents = 0
```

[18]:
```
data_dict = pd.read_csv("data-dictionary.csv")
data_dict.head()
```

[18]:
```
            sex           age_group           variable_group       variables  \
0  Male Female       5 to 17 years  Language Spoken At Home  S1603_C02_002E
1  Male Female      18 to 64 years  Language Spoken At Home  S1603_C02_003E
2  Male Female  65 years and over   Language Spoken At Home  S1603_C02_004E
3  Male Female       5 to 17 years  Language Spoken At Home  S1601_C01_005E
4  Male Female      18 to 64 years  Language Spoken At Home  S1601_C01_006E

                variable_name                    age_category
0  Speak Only English at Home                 child, teenager
1  Speak Only English at Home  young adult, middle aged adult
2  Speak Only English at Home                             old
3                     Spanish                 child, teenager
4                     Spanish  young adult, middle aged adult
```

[19]:
```
def user_selection():
    global selected_persona, selected_percentile, selection_filter,␣
↪variable_inputs

    selected_percentile = widgets.SelectionSlider(
                            options = ['0', '10%', '20%', '30%', '40%', '50%',␣
↪'60%', '70%', '80%', '90%', '100%'],
                            value='50%', description='Percentile',
                            layout=Layout(width='60%', margin='30px 0 30px 0')
                        )

    selected_persona = widgets.ToggleButtons(
                            options=['Boy', 'Teenage Boy', 'Young Man', 'Middle␣
↪Aged Man', 'Old Man',
                                     'Girl', 'Teenage Girl', 'Young Woman', 'Middle␣
↪Aged Woman', 'Old Woman'],
                            value = 'Young Woman', description='AGE:',
                            layout=Layout(width='95%')
                        )

    if (selected_persona.value == 'Boy'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('child')) & \
                                (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Girl'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('child')) & \
                                (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Teenage Boy'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('teenager')) & \
                                (data_dict.sex.str.contains('Male'))]
```

```
    elif (selected_persona.value == 'Teenage Girl'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('teenager')) & \
                                (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Young Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('young')) & \
                            (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Young Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('young')) & \
                            (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Middle Aged Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('middle aged')) & \
                            (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Middle Aged Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('middle aged')) & \
                            (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Old Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('old')) & \
                                (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Old Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('old')) & \
                                (data_dict.sex.str.contains('Female'))]

    list_of_variable_inputs = selection_filter["variables"].values[0:]
    variable_inputs = ', '.join(list_of_variable_inputs).replace(" ", "")
    variable_inputs = variable_inputs.split(',')

user_selection()
```

[20]:
```
def selection_filtering(age_category, sex):

    if (selected_persona.value == 'Boy'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('child')) & \
                                (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Girl'):
```

```
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('child')) & \
                                (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Teenage Boy'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('teenager')) & \
                                (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Teenage Girl'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('teenager')) & \
                                (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Young Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('young')) & \
                            (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Young Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('young')) & \
                            (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Middle Aged Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('middle aged')) & \
                            (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Middle Aged Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('middle aged')) & \
                            (data_dict.sex.str.contains('Female'))]

    elif (selected_persona.value == 'Old Man'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('old')) & \
                                (data_dict.sex.str.contains('Male'))]
    elif (selected_persona.value == 'Old Woman'):
        selection_filter = data_dict[(data_dict.age_category.str.
↪contains('old')) & \
                                (data_dict.sex.str.contains('Female'))]

    list_of_variable_inputs = selection_filter["variables"].values[0:]
    variable_inputs = ', '.join(list_of_variable_inputs).replace(" ", "")
    variable_inputs = variable_inputs.split(',')

    data_output.clear_output()
    output.clear_output()
```

```
def selected_persona_eventhandler(change):
    selection_filtering(change.new, selected_persona.value)
def selected_percentile_eventhandler(change):
    selection_filtering(selected_percentile.value, change.new)
```

[21]:
```
def get_demographics_for_selection():

    global percentile_input, data

    data = pd.merge(acs_site_sum.loc[acs_site_sum['variables'].
 ↪isin(variable_inputs)], \
                    selection_filter, how="outer", on="variables")
    data["sum_in_area"] = data["sum_in_area"].astype(int)
    data.sort_values("sum_in_area", axis = 0, ascending = True, inplace = True)

    percentile_input = int(selected_percentile.value.strip('%')) / 100

# split these up into the diff bins for different types of variable groups
    global language, education, family_household_income,␣
 ↪nonfamily_household_income, household_type

    for item,i in enumerate(data):
        language = data[(data["variable_group"].str.contains('Language'))]
        education = data[(data["variable_group"].str.contains('Educational␣
 ↪Attainment'))]
        family_household_income = data[(data["variable_group"].str.
 ↪contains('Family'))]
        nonfamily_household_income = data[(data["variable_group"].str.
 ↪contains('Nonfamily'))]
        household_type = data[(data["variable_group"].str.
 ↪contains('Households'))]

#Calculate individual percentile values
        global sum_for_percentile_language,\
                sum_for_percentile_education,\
                sum_for_percentile_family_household_income,\
                sum_for_percentile_nonfamily_household_income, \
                sum_for_percentile_household_type

        sum_for_percentile_language = language.sum_in_area.
 ↪quantile(percentile_input).astype(str)
        sum_for_percentile_language = sum_for_percentile_language.
 ↪replace(sum_for_percentile_language, \
                            language.sum_in_area.quantile(percentile_input).
 ↪astype(str))
```

```
        sum_for_percentile_education = education.sum_in_area.
↪quantile(percentile_input).astype(str)
        sum_for_percentile_education = sum_for_percentile_education.
↪replace(sum_for_percentile_education, \
                            education.sum_in_area.quantile(percentile_input).
↪astype(str))

        sum_for_percentile_family_household_income = family_household_income.
↪sum_in_area.quantile(percentile_input).astype(str)
        sum_for_percentile_family_household_income =␣
↪sum_for_percentile_family_household_income.
↪replace(sum_for_percentile_family_household_income, \
                            family_household_income.sum_in_area.
↪quantile(percentile_input).astype(str))

        sum_for_percentile_nonfamily_household_income =␣
↪nonfamily_household_income.sum_in_area.quantile(percentile_input).astype(str)
        sum_for_percentile_nonfamily_household_income =␣
↪sum_for_percentile_nonfamily_household_income.
↪replace(sum_for_percentile_nonfamily_household_income, \
                            nonfamily_household_income.sum_in_area.
↪quantile(percentile_input).astype(str))

        sum_for_percentile_household_type = household_type.sum_in_area.
↪quantile(percentile_input).astype(str)
        sum_for_percentile_household_type = sum_for_percentile_household_type.
↪replace(sum_for_percentile_household_type, \
                            household_type.sum_in_area.
↪quantile(percentile_input).astype(str))
```

[22]:
```
def parse_tables_for_percentile_value():
#generating new transposed table with only the two fields needed : variables␣
 ↪and sum in area.

    global household_type_transposed, language_transposed,␣
 ↪education_transposed, family_household_income_transposed,␣
 ↪nonfamily_household_income_transposed,household_type_transposed

    language_transposed = language.filter(["variables", "sum_in_area"]).T
    language_transposed.columns = language_transposed.iloc[0]
    language_transposed = language_transposed[1:]

    education_transposed = education.filter(["variables", "sum_in_area"]).T
    education_transposed.columns = education_transposed.iloc[0]
    education_transposed = education_transposed[1:]
```

```
    household_type_transposed = household_type.filter(["variables",␣
↪"sum_in_area"]).T
    household_type_transposed.columns = household_type_transposed.iloc[0]
    household_type_transposed = household_type_transposed[1:]

    family_household_income_transposed = family_household_income.
↪filter(["variables", "sum_in_area"]).T
    family_household_income_transposed.columns =␣
↪family_household_income_transposed.iloc[0]
    family_household_income_transposed = family_household_income_transposed[1:]

    nonfamily_household_income_transposed = nonfamily_household_income.
↪filter(["variables", "sum_in_area"]).T
    nonfamily_household_income_transposed.columns =␣
↪nonfamily_household_income_transposed.iloc[0]
    nonfamily_household_income_transposed =␣
↪nonfamily_household_income_transposed[1:]
```

[23]:
```
def get_range_for_each_variable():

    global range_table, range_table_all, ranges, first_range, other_ranges

    data.sort_values(by=['variable_group', 'sum_in_area'], ascending=[True,␣
↪True], inplace=True)
    data_sorted = data.reset_index()

    ranges=[]
    transposed = [education_transposed, family_household_income_transposed,␣
↪household_type_transposed, \
            language_transposed, nonfamily_household_income_transposed]

    for df in transposed:
        for item, i in enumerate(df.columns):
            if item == 0:
                first_range = np.arange(df.max()[item]+1).astype(int)
                ranges.append([first_range])
            else:
                other_ranges = np.arange(df.min()[item-1]+1, \
                                         df.max()[item]+1).astype(int)
                ranges.append([other_ranges])

            range_table = pd.DataFrame(data=ranges, index=None,␣
↪columns=["range_per_variable"])
            range_table = range_table.reset_index(drop=True)
```

```
    range_table_all = pd.merge(range_table, data_sorted, left_index=True,␣
↪right_index=True, on=None)
    range_table_all["range_per_variable"] =␣
↪range_table_all["range_per_variable"].astype(str)
```

[24]:
```
def generate_info_for_text():

    global result_df

    result_df = pd.DataFrame(columns=None)
    for i in range_table_all['range_per_variable']:
        if '\n' in range_table_all:
            range_table_all['range_per_variable'].replace(r'\s+|\\n', ' ',␣
↪regex=True, inplace=True)

    sum_for_percentile_language = int(language.sum_in_area.
↪quantile(percentile_input))
    sum_for_percentile_education = education.sum_in_area.
↪quantile(percentile_input)
    if (sum_for_percentile_education == 'nan'):
        sum_for_percentile_education = 0
    elif (sum_for_percentile_education != 'nan') &␣
↪(sum_for_percentile_education != 0):
        sum_for_percentile_education = int(sum_for_percentile_education)
    sum_for_percentile_family_household_income = int(family_household_income.
↪sum_in_area.quantile(percentile_input))
    sum_for_percentile_nonfamily_household_income =␣
↪int(nonfamily_household_income.sum_in_area.quantile(percentile_input))
    sum_for_percentile_household_type = int(household_type.sum_in_area.
↪quantile(percentile_input))

    for item,i in enumerate(range_table_all.index):

        if int(sum_for_percentile_language) > 0 :
            language_only = range_table_all[(range_table_all["variable_group"].
↪str.contains('Language'))]
            result = language_only[language_only["range_per_variable"].str.
↪contains(str(sum_for_percentile_language))]
            result_df = result_df.append(result, ignore_index = True)

        if int(sum_for_percentile_education > 0):
            education_only = range_table_all[(range_table_all["variable_group"].
↪str.contains('Educational'))]
            result = education_only[education_only["range_per_variable"].str.
↪contains(str(sum_for_percentile_education))]
            result_df = result_df.append(result, ignore_index = True)
```

```
        if int(sum_for_percentile_family_household_income) > 0:
            family_household_income_only =␣
↪range_table_all[(range_table_all["variable_group"].str.contains('Family'))]
            result =␣
↪family_household_income_only[family_household_income_only["range_per_variable"].
↪str.contains(str(sum_for_percentile_family_household_income))]
            result_df = result_df.append(result, ignore_index = True)

        if int(sum_for_percentile_nonfamily_household_income) > 0:
            nonfamily_household_income_only =␣
↪range_table_all[(range_table_all["variable_group"].str.
↪contains('Nonfamily'))]
            result =␣
↪nonfamily_household_income_only[nonfamily_household_income_only["range_per_variable"].
↪str.contains(str(sum_for_percentile_nonfamily_household_income))]
            result_df = result_df.append(result, ignore_index = True)

        if int(sum_for_percentile_household_type) > 0:
            household_type_only =␣
↪range_table_all[(range_table_all["variable_group"].str.
↪contains('Households'))]
            result =␣
↪household_type_only[household_type_only["range_per_variable"].str.
↪contains(str(sum_for_percentile_household_type))]
            result_df = result_df.append(result, ignore_index = True)

    result_df = result_df.drop_duplicates()
```

[25]:
```
def import_google_data():
    extract_location()

    keys_file = open("gcs_key.txt")
    APIKEY = keys_file.read().strip()

    global total_results
    total_results = []
    types = ["bar","cafe","restaurant"]

    for i in types:
        def findPlaces(pagetoken = None):
            global lat, lon
            lat, lng = (lat,lon)
            radius=402
```

```
            url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?
↪location={lat},{lng}&radius={radius}&fields=name,geometry,types,price_level&type={type}&key
↪format(lat = lat, lng = lng, radius = radius, type = type,APIKEY = APIKEY,␣
↪pagetoken = "&pagetoken="+pagetoken if pagetoken else "")
            response = requests.get(url)
            res = json.loads(response.text)

            for result in res["results"]:
                place_name = result['name']
                latitude = result["geometry"]["location"]["lat"]
                longitude = result["geometry"]["location"]["lng"]
                place_type = result.get("types",0)
                price_level = result.get("price_level",0)
                total_results.append([latitude, longitude,␣
↪place_name,place_type,price_level])

            pagetoken = res.get("next_page_token",None)

            return pagetoken

        pagetoken = None

        while True:
             pagetoken = findPlaces(pagetoken=pagetoken)
             import time
             time.sleep(5)

             if not pagetoken:
                 break
```

[26]:
```
def generate_google_data_for_text():
    import_google_data()

    global places_df, establishment, price_range, list_ranges, very_expensive,␣
↪expensive, moderately_priced, cheap, num_very_expensive, num_expensive,␣
↪num_moderately_priced, num_cheap
    places_df = pd.DataFrame(data=total_results, columns =["latitude",␣
↪"longitude", "place_name","place_type","price_level"])

    very_expensive = places_df[((places_df["price_level"]).astype(str).str.
↪contains('4'))]
    expensive = places_df[((places_df["price_level"]).astype(str).str.
↪contains('3'))]
    moderately_priced = places_df[((places_df["price_level"]).astype(str).str.
↪contains('2'))]
```

```
    cheap = places_df[((places_df["price_level"]).astype(str).str.
↪contains('1'))]

    num_very_expensive = len(very_expensive)
    num_expensive = len(expensive)
    num_moderately_priced = len(moderately_priced)
    num_cheap = len(cheap)

    list_ranges = [num_very_expensive, num_expensive, num_moderately_priced,␣
↪num_cheap]

    for i in places_df['price_level']:
        if max(list_ranges) == num_very_expensive:
            price_range = ", are very expensive."
            establishment = random.choice(very_expensive["place_name"].values)

        if max(list_ranges) == num_expensive:
            price_range = ", are expensive."
            establishment = random.choice(expensive["place_name"].values)

        if max(list_ranges) == num_moderately_priced:
            price_range = ", are moderately expensive."
            establishment = random.choice(moderately_priced["place_name"].
↪values)

        if max(list_ranges) == num_cheap:
            price_range = ", are quite cheap."
            establishment = random.choice(cheap["place_name"].values)
```

[27]:
```
def construct_narrative():

    clean_combine_census_and_geographic_data()
    determine_population_distribution()

    selected_persona.observe(selected_persona_eventhandler, names='value')
    selected_percentile.observe(selected_percentile_eventhandler, names='value')

    get_demographics_for_selection()
    parse_tables_for_percentile_value()
    get_range_for_each_variable()
    generate_info_for_text()
    generate_google_data_for_text()

    global result_df, resident_text, \
        intro_text, percentile_text, household_descriptor, establishments_text,␣
↪affordability_text, \
```

```
      household_type_text, lanpguage_text, language_adjective,␣
↪education_text,\
      income_text, income_range, \
      establishment, price_range, price_experience, price_marker,␣
↪affordability_descriptor, comfort_descriptor,\
      area_type, persona_type

   greetings=["Hi, ", "Hey, ", "Hello, "]
   education_text = ''
   persona_type = selected_persona.value.lower()

   if more_residents >= 50:
       area_type = "predominantly residential area. "
   elif (more_residents > 0) & (more_residents < 50):
       area_type = "residential area. "
   elif more_workers > 50 :
       area_type = "predominantly commercial area. "
   elif (more_workers > 0) & (more_workers < 50):
       area_type = "commercial area. "
   elif resident_percent == worker_percent:
       area_type = "area that is both commercial and residential. "

   for i in result_df['variable_name']:
       if 'Less than high' in i:
           education_text = " I never got my high school diploma."
       if 'High school graduate or higher' in i:
           education_text = 'I am a high school graduate. '
       if 'Some college' in i:
           education_text = " While I've attended some form of college, I␣
↪never completed my degree."
       if 'Bachelor' in i:
           education_text = " I have a Bachelor's degree, and might even have␣
↪attained professional or other advanced degrees."

       if 'English' in i:
           language_text = "where English is the only language spoken. "
           language_adjective = ""
       if 'Spanish' in i:
           language_text = "where we speak both English and Spanish. "
       if 'Indo-European' in i:
           language_text = "where we speak both English and an Indo-European␣
↪language. "
       if 'Asian' in i:
           language_text = "where we speak both English and an Asian or␣
↪Pacific Island language. "
       if 'Other languages' in i:
```

```
            language_text = "where we speak another language in addition to␣
↪English. "
        else:
            language_adjective = " bilingual "

    #Removing family income from dataframe if it is a non-family household and␣
↪vice versa
        if 'Nonfamily households' in i:
            household_type_text = " away from my family "
            result_df=result_df.loc[~result_df["variable_name"].str.
↪contains('Family Household Income')]
        elif 'Family households' in i:
            household_type_text = ' with my family '
            result_df=result_df.loc[~result_df["variable_name"].str.
↪contains('Nonfamily Household Income')]
        else:
            household_type_text=' household '


        if '10,000' or '14,999' in i:
            income_text = " lower income"
            for i in places_df['price_level']:
                if (max(list_ranges) == num_very_expensive) | (max(list_ranges)␣
↪== num_expensive):
                    price_marker = "prohibitively expensive"
                    affordability_descriptor = "this is a big issue"
                    comfort_descriptor = "not"
                elif max(list_ranges) == num_moderately_priced:
                    price_marker = "expensive"
                    affordability_descriptor = "this is an issue"
                    comfort_descriptor = "barely"
                elif max(list_ranges) == num_cheap:
                    price_marker = "affordable"
                    affordability_descriptor = "this is not an issue"
                    comfort_descriptor = "comfortably"
        if '24,999' or '34,999' or '49,999' in i:
            income_text = " middle income"
            for i in places_df['price_level']:
                if max(list_ranges) == num_very_expensive:
                    price_marker = "prohibitively expensive"
                    affordability_descriptor = "this is a big issue"
                    comfort_descriptor = "not"
                elif max(list_ranges) == num_expensive:
                    price_marker = "expensive"
                    affordability_descriptor = "this is an issue"
                    comfort_descriptor = "barely"
```

```
                elif (max(list_ranges) == num_cheap) | (max(list_ranges) ==␣
↪num_moderately_priced):
                    price_marker = "affordable"
                    affordability_descriptor = "this is not an issue"
                    comfort_descriptor = "comfortably"
        if '74,999' or '94,999' or '149,999' in i:
            income_text = " wealthy"
            for i in places_df['price_level']:
                if max(list_ranges) == num_very_expensive:
                    price_marker = "expensive"
                    affordability_descriptor = "this is an issue"
                    comfort_descriptor = "barely"
                elif (max(list_ranges) == num_cheap) | (max(list_ranges) ==␣
↪num_moderately_priced) | (max(list_ranges) == num_expensive):
                    price_marker = "affordable"
                    affordability_descriptor = "this is not an issue"
        if '199,999' or '200,000' in i:
            income_text = " very wealthy"
            price_marker = "affordable"
            affordability_descriptor = "this is not an issue"
            comfort_descriptor = "comfortably"


    intro_text = random.choice(greetings) + "I'm a " + persona_type + " living␣
↪in a " + area_type
    percentile_text = str(selected_percentile.value) + " of my neighbors are␣
↪like me. "
    household_descriptor = "I live " + household_type_text + " in a " +␣
↪income_text + language_adjective + " household " + language_text
    establishments_text = " Most of the eating and drinking establishments in␣
↪this area, such as " + establishment + price_range
    affordability_text = " Because I am " + income_text + ", " +␣
↪affordability_descriptor + " as I can " + comfort_descriptor + " afford my␣
↪neighborhood's bars, cafes and restaurants."

    resident_text = intro_text + percentile_text + education_text +␣
↪household_descriptor + establishments_text + affordability_text

    with output:
        display(resident_text)

construct_narrative()
```

[28]:
```
text_generation_button = Button(description="SEE STORY",\
                                layout=Layout(width='757px', height='50px',␣
↪border='solid .5px #000', margin='0 0 0 90px'))
```

```
text_generation_button.style.button_color = '#EDF9FC'
text_generation_button.style.font_weight = 'bold'
```

[29]:
```
def text_generation(b):
    output.clear_output()
    data_output.clear_output()
    construct_narrative()
    show_dashboard()

text_generation_button.on_click(text_generation)
```

[34]:
```
def show_dashboard():
    output.clear_output()
    data_output.clear_output()

    item_layout = widgets.Layout(margin='0 0 10px 0', align_items='stretch')
    item_layout_tab = widgets.Layout(margin='0 0 10px 0')

    explore_data = range_table_all
    explore_data['sum_in_area'] = explore_data['sum_in_area'].astype(int)

    with output:
        display(md("> <font size = 3, font color = black> {}".
 ↪format(resident_text)))
    with data_output:
        display(explore_data)

    global tab, input_widgets
    input_widgets = widgets.VBox(
        [selected_persona, selected_percentile, text_generation_button],
        layout=item_layout)

    tab = widgets.Tab([output, data_output],
        layout=item_layout_tab)
    tab.set_title(0, 'Narrative')
    tab.set_title(1, 'Data')

    global dashboard
    dashboard = widgets.VBox([input_widgets, tab])

show_dashboard()


display(dashboard)
m
```

```
VBox(children=(VBox(children=(ToggleButtons(description='AGE:', index=4, layout=Layout(width='
```

```
Map(center=[40.7210907, -73.9877836], controls=(ZoomControl(options=['position', 'zoom_in_text
```

[ ]: